Trabajo patrones de diseño

Martín López de Ipiña, Borja Gómez y Alex Rivas

Introducción:

En este documento se presenta la implementación de tres patrones de diseño a la práctica *bets*.

1. Patrón Factory Method (Martín Lopez de Ipiña Muñoz)

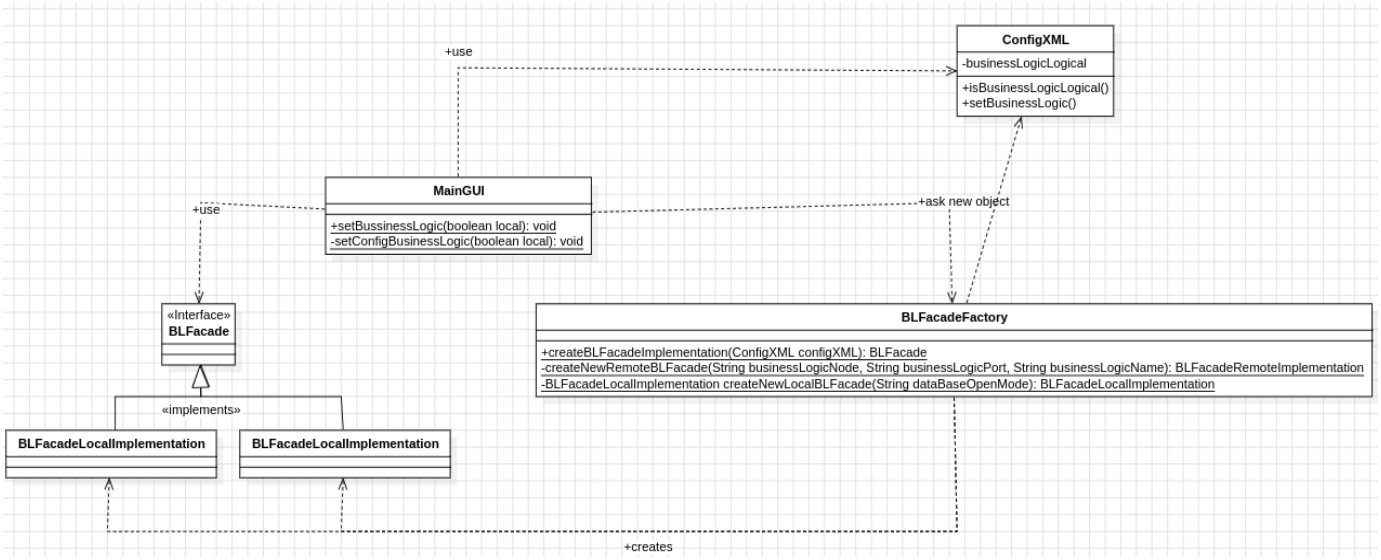
Partiendo desde el siguiente código:

```
public class ApplicationLauncher {
    sarag *
    public static void main(String[] args) {

        ConfigXML c=ConfigXML.getInstance();

        System.out.println(c.getLocale());

        Locale.setDefault(new Locale(c.getLocale()));

        System.out.println("Locale: "+Locale.getDefault());

        MainGUI a=new MainGUI();
        a.setVisible(true);
```

```
try {

    BLFacade appFacadeInterface;
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

    if (c.isBusinessLogicLocal()) {

        //In this option the DataAccess is created by FacadeImplementationWS
        //appFacadeInterface=new BLFacadeImplementation();

        //In this option, you can parameterize the DataAccess (e.g. a Mock DataAccess object)

        DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
        appFacadeInterface=new BLFacadeImplementation(da);

    }else { //If remote

         String serviceName= "http://"+c.getBusinessLogicNode() +":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";

        //URL url = new URL("http://localhost:9999/ws/ruralHouses?wsdl");
        URL url = new URL(serviceName);


        //1st argument refers to wsdl document above
        //2nd argument is service name, refer to wsdl document above
        QName qname = new QName( namespaceURI: "http://businessLogic/",  localPart: "BLFacadeImplementationService");

        Service service = Service.create(url, qname);

        appFacadeInterface = service.getPort(BLFacade.class);
    }
    MainGUI.setBussinessLogic(appFacadeInterface);

}catch (Exception e) {
    a.jLabelSelectOption.setText("Error: "+e.toString());
    a.jLabelSelectOption.setForeground(Color.RED);

    System.out.println("Error in ApplicationLauncher: "+e.toString());
}
```

Toda la fase de creación del BLFacade se realiza en ApplicationLauncher.

He creado el siguiente patrón Factory para segregar la fase de creación e introducirla en la clase MainGUI de la vista:



De esta manera MainGUI se encarga de seleccionar la implementación correspondiente llamando a BLFacadeFactory para crearla. MainGUI especifica en ConfigXML el tipo de implementación correspondiente y mediante el Factory asigna el tipo de implementación.

Si se le pasa el parámetro local a true a la función setBusinessLogic de MainGUI entonces se usará una implementación local, sino se usará una implementación remota con la configuración de ConfigXML.

Además, he movido todo el código necesario para la creación de la implementación correspondiente a su función privada correspondiente en la clase Factory.

- BLFacadeFactory (clase Creator):

```java
public class BLFacadeFactory {

    1 usage  new *
    public static BLFacade createBLFacadeImplementation(ConfigXML configXML) {
        if (configXML.isBusinessLogicLocal()) {
            return createNewLocalBLFacade(configXML.getDataBaseOpenMode());
        } else {
            return createNewRemoteBLFacade(configXML.getBusinessLogicNode(), configXML.getBusinessLogicPort(), configXML.getBusinessLogicName());
        }
    }
```

```java
private static BlFacadeRemoteImplementation createNewRemoteBLFacade(String businessLogicNode, String businessLogicPort, String businessLogicName){

    //example: URL url = new URL("http://localhost:9999/ws/ruralHouses?wsdl");
    String serviceName= "http://"+businessLogicNode +":"+ businessLogicPort+"/ws/"+businessLogicName+"?wsdl";

    URL url = null;
    try {
        url = new URL(serviceName);
    } catch (MalformedURLException e) {
        System.out.println("Error in ApplicationLauncher: "+e.toString());
        throw new RuntimeException(e);
    }


    //1st argument refers to wsdl document above and 2nd argument is service name, refer to wsdl document above
    QName qname = new QName( namespaceURI: "http://businessLogic/",  localPart: "BLFacadeImplementationService");

    Service service = Service.create(url, qname);

    return new BlFacadeRemoteImplementation(service);
}


//In this option, you can parameterize the DataAccess (e.g. a Mock DataAccess object)
1 usage  new *
private static BLFacadeLocalImplementation createNewLocalBLFacade(String dataBaseOpenMode) {

    DataAccess da = new DataAccess(dataBaseOpenMode.equals("initialize"));
    return new BLFacadeLocalImplementation(da);
}
```

- BLFacadeRemoteImplementation (clase concreteProduct): He renombrado BLFacadeImplementation a BLFacadeLocalImplementation para poder crear la clase BLFacadeRemoteImplementation. La clase BLFacadeRemoteImplementation se encarga de implementar BLFacade en los casos en la que la conexión a la base de datos es remota. Contiene un objeto BLFacade creado mediante el service.getPort() (la misma implementación de antes) que se encarga de implementar los métodos de BLFacade:

```java
public class BlFacadeRemoteImplementation implements BLFacade{

    24 usages
    BLFacade blf;

    1 usage  new *
    public BlFacadeRemoteImplementation(Service service){
        blf = service.getPort(BLFacade.class);
    }

    new *
    @Override
    public Question createQuestion(Event event, String question, float betMinimum) throws EventFinished, QuestionAlreadyExist {
        return blf.createQuestion(event, question, betMinimum);
    }

    new *
    @Override
    public Vector<Event> getEvents(Date date) {
        return blf.getEvents(date);
    }

    new *
    @Override
    public Vector<Date> getEventsMonth(Date date) {
        return blf.getEventsMonth(date);
    }

    1 usage  new *
    @Override
    public void initializeBD() {
        blf.initializeBD();
    }
```

```java
@Override
public User createUser(User user) throws UserAlreadyExist {
    return blf.createUser(user);
}

new *
@Override
public Event createEvent(String description, Date eventDate) throws EventAlreadyExist {
    return blf.createEvent(description, eventDate);
}

new *
@Override
public Forecast createForecast(String description, float gain, int questionNumber) throws ForecastAlreadyExist, QuestionDoesntExist {
    return blf.createForecast(description, gain, questionNumber);
}

new *
@Override
public User getUser(String Dni) throws UserDoesntExist {
    return blf.getUser(Dni);
}

new *
@Override
public Question getQuestion(Integer questionNumber) throws QuestionDoesntExist {
    return blf.getQuestion(questionNumber);
}

new *
@Override
public void assignResult(Integer questionNumber, Integer forecastNumber) throws QuestionDoesntExist, ForecastDoesntExist, EventHasntFinished {
    blf.assignResult(questionNumber, forecastNumber);
}
```

```java
@Override
public void removeBet(Integer betNumber) throws BetDoesntExist {
    blf.removeBet(betNumber);
}

new *
@Override
public Bet modifyBet(float betMoney, int betNumber, String dni) throws BetDoesntExist, UserDoesntExist {
    return blf.modifyBet(betMoney, betNumber, dni);
}

new *
@Override
public void modifyUserName(User user, String Nombre2) {
    blf.modifyUserName(user, Nombre2);
}

new *
@Override
public void modifyUserApellido(User user, String Apellido) {
    blf.modifyUserApellido(user, Apellido);
}

new *
@Override
public void modifyUserUsuario(User user, String Usuario) {
    blf.modifyUserUsuario(user, Usuario);
}

new *
@Override
public void modifyUserPasswd(User user, String passwd) {
    blf.modifyUserPasswd(user, passwd);
}

new *
@Override
public void modifyUserCreditCard(String user, Long newCard) {
    blf.modifyUserCreditCard(user, newCard);
}
```

```java
@Override
public Vector<User> getAllUsers() {
    return blf.getAllUsers();
}

new *
@Override
public boolean removeUser(String dni) {
    return blf.removeUser(dni);
}

new *
@Override
public Bet createBet(String dni, float betMoney, int forecastNumber) throws BetAlreadyExist, UserDoesntExist, ForecastDoesntExist {
    return blf.createBet(dni, betMoney, forecastNumber);
}

new *
@Override
public User modifySaldo(float saldo, String user2) {
    return blf.modifySaldo(saldo, user2);
}

new *
@Override
public Forecast getForecast(Integer forecastNumber) throws ForecastDoesntExist {
    return blf.getForecast(forecastNumber);
}

2 usages  new *
@Override
public void updateCloseEvent(Integer numResultado) {
    blf.updateCloseEvent(numResultado);
}
```

- BLFacadeLocalImplementation (clase ConcreteProduct): Se encarga de implementar BLFacade en los casos en los que la conexión es local. Sin cambios respecto a BLFacadeImplementation.

- MainGUI (clase que llama al factory):

Se encarga de elegir la implementación de lógica de negocio:

```java
public static void setBussinessLogic(boolean local){
    setConfigBusinessLogic(local);
    appFacadeInterface= BLFacadeFactory.createBLFacadeImplementation(ConfigXML.getInstance());
}

1 usage  new *
private static void setConfigBusinessLogic(boolean local) {
    ConfigXML.getInstance().setBusinessLogic(local);
}
```

- ConfigXML:

Se encarga de guardar la configuración de la lógica de negocio y algunos parámetros necesarios para la conexión remota:

```java
public boolean isBusinessLogicLocal() { return businessLogicLocal; }

new *
public void setBusinessLogic(boolean local) {
    businessLogicLocal=local;
}
```

- AplicationLauncher:

Mediante la clase MainGUI pone la lógica de negocio a local como predeterminado:

```java
public class ApplicationLauncher {
    sarag *
    public static void main(String[] args) {

        ConfigXML c=ConfigXML.getInstance();
        System.out.println(c.getLocale());

        Locale.setDefault(new Locale(c.getLocale()));
        System.out.println("Locale: "+Locale.getDefault());

        MainGUI a=new MainGUI();
        a.setVisible(true);

        try {

            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

            MainGUI.setBussinessLogic(true);

        }catch (Exception e) {
            a.jLabelSelectOption.setText("Error: "+e.toString());
            a.jLabelSelectOption.setForeground(Color.RED);

            System.out.println("Error in ApplicationLauncher: "+e.toString());
        }
    }
}
```

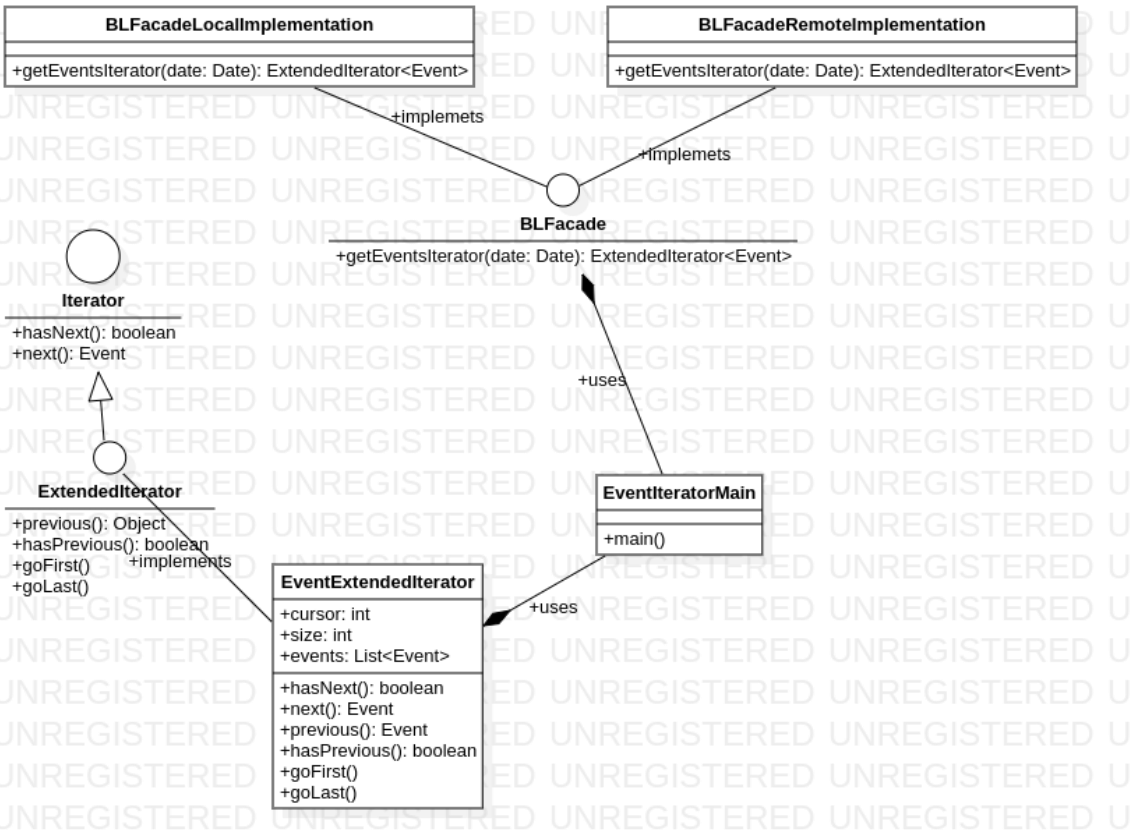
2. Patrón Iterator (Borja Gómez Calvo)

Mediante el patrón iterator queremos dar al cliente una forma distinta de iterar los eventos. Para ello hemos creado una interfaz nueva, ExtendedIterator, la cual añade nuevos metodos a la interfaz padre Iterator para poder iterar las estructuras de datos hacia adelante y detrás con el mismo iterador.

Una vez tenemos las interfaces especificadas, crearemos una implementación de dicha interfaz en una lista de eventos. Esta clase se llamará EventsExtendedIterator.

La interfaz BLFacade implementará el método getEventsIterator, el cual devolverá una instancia de la interfaz ExtendedIterator<Events> para poder hacer uso de esta nueva implementación.

Diagrama UML que muestra la relación de clases:



A continuación, los cambios en el código:

BLFacade:

```
ExtendedIterator<Event> getEventsIterator(Date date);
```

BLFacadeLocalImplementation:

```
@Override
public ExtendedIterator<Event> getEventsIterator(Date date) {
   List<Event> events = getEvents(date);
```

```
    return new EventExtendedIterator(events);
}
```

`BLFacadeRemoteImplementation:`

```
@Override
public ExtendedIterator<Event> getEventsIterator(Date date) {
    return blf.getEventsIterator(date);
}
```

`ExtendedIterator:`

```
package iterators;

import java.util.Iterator;

public interface ExtendedIterator<Object> extends Iterator<Object> {
    public Object previous();
    public boolean hasPrevious();
    public void goFirst();
    public void goLast();

}
```

`EventExtendedIterator:`

```
package iterators;

import domain.Event;

import java.util.List;
import java.util.NoSuchElementException;

public class EventExtendedIterator implements ExtendedIterator<Event> {

    private int cursor = -1;
    private final int size;
    private final List<Event> events;

    public EventExtendedIterator(List<Event> events) {
        this.events = events;
        this.size = events.size()-1;
    }

    @Override
    public boolean hasNext() {
        return cursor != size;
    }

    @Override
    public Event next() {
        int i = cursor;
        if (i >= size)
            throw new NoSuchElementException();
```

```
    cursor = i + 1;
    return events.get(cursor);
  }

  @Override
  public Event previous() {
    int i = cursor;
    if (i >= size && i<=0)
      throw new NoSuchElementException();
    cursor = i - 1;
    return events.get(cursor);
  }

  @Override
  public boolean hasPrevious() {
    return cursor != 0;
  }

  @Override
  public void goFirst() {
    cursor = -1;
  }

  @Override
  public void goLast() {
    cursor = size+1;
  }
}
```

`EventIteratorMain:`

```
package iterators;

import businessLogic.BLFacade;
import businessLogic.BLFacadeFactory;
import configuration.ConfigXML;
import domain.Event;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class EventIteratorMain {

  public static void main(String[] args) {
    // obtener el objeto Facade local
    ConfigXML configXML = ConfigXML.getInstance();
    // establecemos la logica de negocio a local
    configXML.setBusinessLogic(true);
    BLFacade blFacade = BLFacadeFactory.createBLFacadeImplementation(configXML);
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    Date date;
    try {
      date = sdf.parse("17/05/2023"); // 17 del mes que viene
```

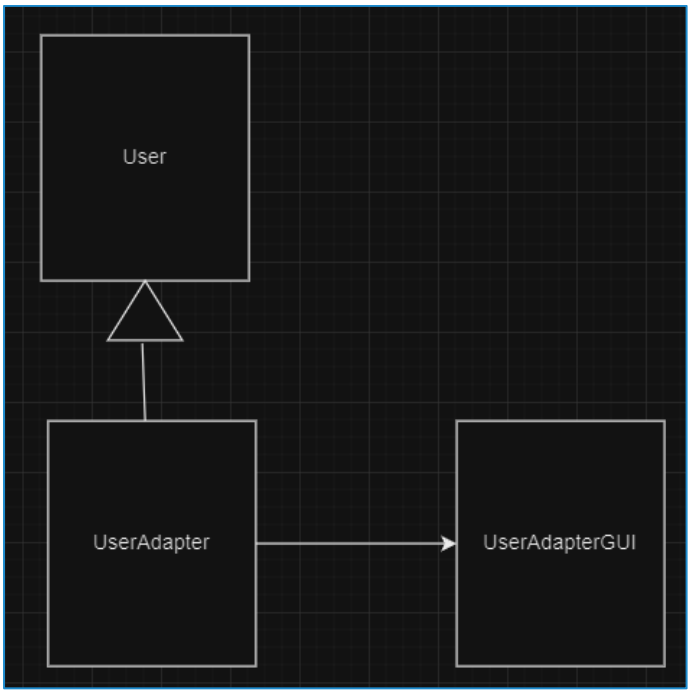
```
        ExtendedIterator<Event> i = blFacade.getEventsIterator(date);
        Event e;
        System.out.println("_____");
        System.out.println("RECORRIDO HACIA ATRÁS");
        i.goLast(); // Hacia atrás
        while (i.hasPrevious()) {
          e = i.previous();
          System.out.println(e.toString());
        }
        System.out.println();
        System.out.println("_____");
        System.out.println("RECORRIDO HACIA ADELANTE");
        i.goFirst(); // Hacia adelante
        while (i.hasNext()) {
          e = i.next();
          System.out.println(e.toString());
        }
      } catch (ParseException e1) {
        System.out.println("Problems with date?? " + "17/12/2020");
      }
    }
}
```

Por último, una prueba de la ejecución:

```
_____
RECORRIDO HACIA ATRÁS
10;Betis-Real Madrid
9;Real Sociedad-Levante
8;Girona-Leganés
7;Malaga-Valencia
6;Las Palmas-Sevilla
5;Espanyol-Villarreal
4;Alavés-Deportivo La Coruña
3;Getafe-Celta De Vigo
2;Eibar-Barcelona
1;Atlético-Athletic

_____
RECORRIDO HACIA ADELANTE
1;Atlético-Athletic
2;Eibar-Barcelona
3;Getafe-Celta De Vigo
4;Alavés-Deportivo La Coruña
5;Espanyol-Villarreal
6;Las Palmas-Sevilla
7;Malaga-Valencia
8;Girona-Leganés
9;Real Sociedad-Levante
10;Betis-Real Madrid

Process finished with exit code 0
```

3. Patrón Adapter (Alex Rivas Machín)

En este patrón buscamos adaptar los métodos de User para representar los datos de las apuestas que ha realizado en una tabla (JTable).



- Para aplicar el patrón hemos creado dos clases(UserAdapter y UserAdapterGUI). UserAdapter se encarga de la creación de objetos adaptables para la creación de JTables. Por otra parte, UserAdapterGUI se encarga de la creación de Jtables y el JFrame donde la representas.

UserAdapter:

```java
public class UserAdapter extends AbstractTableModel {
    private User user;
    private List<Bet> apuestas;
    AlexSISISI
    public UserAdapter(User user) {
        this.user = user;
        this.apuestas=user.getBets();
    }
    AlexSISISI
    @Override
    public int getRowCount() {
        return apuestas.size();
    }
    AlexSISISI
    @Override
    public int getColumnCount() {
        return 4;
    }
    AlexSISISI
    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Bet bet = apuestas.get(rowIndex);
        switch (columnIndex){
            case 0: return bet.getForecast().getQuestion().getEvent().getDescription();
            case 1:return bet.getForecast().getQuestion().getQuestion();
            case 2:return bet.getForecast().getQuestion().getEvent().getEventDate();
            case 3:return bet.getBetMoney();
            default: return null;
        }
    }

    AlexSISISI
    @Override
    public String getColumnName(int column) {
        return super.getColumnName(column);
    }
}
```

AdapterGUI:

```java
public class AdapterGUI  extends JFrame {
    private User user;
    private JTable  tabla;
    AlexSISISI
    public AdapterGUI(User  user){
        super( title: "Apuesta realizadas por "+user.getName()+":");
        this.setBounds( x: 100,   y: 100,  width: 700,   height: 200);
        this.user = user;
        UserAdapter adapt = new UserAdapter(user);
        tabla = new JTable(adapt);
        tabla.setPreferredScrollableViewportSize(new Dimension( width: 500,   height: 70));
        //Creamos un JscrollPane    y   le agregamos la JTable
        JScrollPane scrollPane =    new JScrollPane(tabla);
        //Agregamos el  JScrollPane al contenedor
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }
}
```

Ejecución:

```java
public class Main {
    AlexSISISI
    public static void main(String[] args) {
        try {
            BLFacadeImplementation blFacade = new BLFacadeImplementation();
            blFacade.getAllUsers().forEach(user → System.out.println(user.getDni()));
            User user = blFacade.getUser( Dni: "123456789N");
            AdapterGUI vt = new AdapterGUI(user);
            vt.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```